RAID 6 Hardware Acceleration

MICHAEL GILROY, JAMES IRVINE, and ROBERT ATKINSON, University of Strathclyde

Inexpensive, reliable hard disk storage is increasingly required in both businesses and the home. As disk capacities increase and multiple drives are combined in one system the probability of multiple disk failures increases. Through the adoption of RAID 6 the capability to recover from up to two simultaneous disk failures becomes available. In this article, we present three different RAID 6 implementations each tailored to support different target applications and optimized to reduce overall hardware resource utilization. We present an optimal Reed-Solomon-based RAID 6 implementation for arrays of four disks. We also present the smallest in terms of hardware resource utilization as well having the highest throughput RAID 6 hardware solution for disk arrays of up to 15 drives. Finally, we present an implementation supporting up to 255 disks in a single array.

Categories and Subject Descriptors: B.3.2. [**Memory Stuctures**]: Design Styles—*Mass storage hardware*

General Terms: Reliability, Design, Verification

Additional Key Words and Phrases: RAID 6, Reed-Solomon codec

ACM Reference Format:
Gilroy, M., Irvine, J., and Atkinson, R. 2011. RAID 6 hardware acceleration. ACM Trans. Embed. Comput. Syst. 10, 4, Article 43 (November 2011), 17 pages.
DOI = 10.1145/2043662.2043667 http://doi.acm.org/10.1145/2043662.2043667

1. INTRODUCTION

The most common solution for the provision of reliable data storage is Redundant Arrays of Independent Disks (RAID) which enable multiple hard disk drives to be combined to offer a combination of better performance and tolerance to failure [Patterson et al. 1988]. RAID-based systems have tended to utilize high-reliability/low-density disk drives.

Economic storage systems must strike a delicate balance between cost, reliability, and capacity. The objective is to obtain the maximum capacity for minimum cost at a given level of reliability.

Traditionally, Small Computer System Interface (SCSI) disk drives were characterized as high-reliability/high-cost devices. By contrast, ATA drives were characterized as low-reliability/low-cost devices. The main solution for reliable RAID storage systems, RAID 5, utilizes parallelism through single disk redundancy, that is, the ability to recover from a single disk failure, to increase the overall reliability of the storage system. Consequently, employing RAID storage systems based upon Advanced Technology Attachment (ATA) disk technology has the potential to provide a more cost-effective alternative to comparable reliable storage systems employing SCSI or SAS

This work was supported by the EPSRC and A2E Ltd.
Authors' address: M. Gilroy (corresponding author), J. Irvine, and R. Atkinson, University of Stathclyde, Glasgow, UK: email: mgilroy@theiet.org.

DOI 10.1145/2043662.2043667 http://doi.acm.org/10.1145/2043662.2043667

technology. The downside of RAID arrays is that the probability of more than one simultaneous disk failure increases with the number of disks in the array.

This represents a significant drawback in the application of low-cost/low-reliability disk drives to storage systems. By using a RAID scheme which provides mechanisms to recover data when up to two disk drives fail simultaneously a reliable, high-capacity storage system may be implemented with lower-cost disk drives.

The rest of this article is structured as follows: Section 2 provides an overview of RAID 5 and RAID 6. Section 3 details various RAID 6 algorithms and implementations. Our initial optimal RAID 6 implementation is described in Section 4 with the reliability and performance of this design discussed in Section 5. Two improved RAID 6 controller supporting arrays of up to 15 and 255 disks are described in Section 6. We then provide an implementation of the RAID 6 controller for the Altera Avalon bus in Section 7 to demonstrate the ease of use of these IP blocks and to support testing of disk arrays of up to 255 disk drives. Finally, Section 8 provides the conclusions and analysis of the article.

2. TECHNICAL BACKGROUND

The use of RAID was first proposed in Patterson et al. [1988] as a means of providing higher-capacity data storage and providing recovery mechanisms in the event of a single disk failure. Single large hard disk drives remain the predominant storage technology despite the known issues of complete data loss resulting from disk failure, or at the very least the need to recover data from a backup system. Backup system costs are expensive so there is a need to make the primary data storage system tolerant to disk failures.

The benefits of RAID include:

— protection against data loss;
— provision of real-time data recovery with uninterrupted access when a disk fails;
— multiple drives working in parallel, increasing overall system performance;
— increased system uptime and availability.

In addition to providing increased reliability, disk arrays improve the data throughput of storage systems [Hennessy and Patterson 2002] as a result of disk parallelism. The disadvantage associated with disk arrays is that as the number of data disks in a system increases the reliability of the system decreases, since more devices results in more chance of a failure. The Mean Time To Failure (MTTF) of a disk array is inversely proportional to the number of devices in an array, as shown in Eq. (1). We have

$$MTTF = \frac{\tau_{fail}}{D}, \tag{1}$$

where τ_{fail} is the mean time to failure of a disk and D is the number of disks in the array.

Adding redundant storage devices to a disk array increases the array's tolerance to faults. Should a disk fail the lost information may be recovered from the redundant information stored on the remaining disks. The lost data can be successfully recovered provided further data losses, exceeding the redundancy built into the array, do not occur during the recovery period, that is, the time before the disk can be replaced and array be rebuilt.

The mean time to failure of modern hard disk drives is over 1 million hours [Maxtor Corporation 2005a, 2005b]. Data loss, however, is not solely dependant upon disk failure; the rate at which nonrecoverable errors occur is about 1 in 10^{15}. When a 15 data disk array with a single parity disk fails data lost in any one of these disk sectors due to a nonrecoverable error will result in the loss of data. Assuming a drive capacity

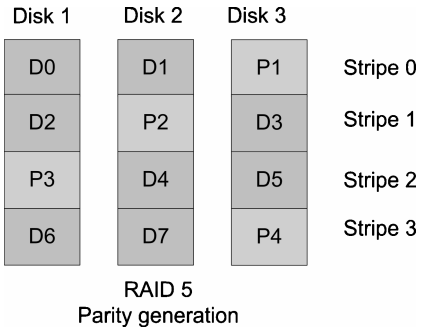

Fig. 1. RAID 5 implementation.

of 250GB the probability of reading all these drives successfully is 0.9758. This means that approximately 2% of all disk failures will result in data loss. Additionally, further disk failures prior to the array recovering will result in a catastrophic loss of data.

This article details only RAID 5 and RAID 6 implementations. A comprehensive analysis of the performance benefits of each of the most common RAID levels is available in Chen et al. [1994].

2.1 RAID 5

RAID level 5 writes blocks of data (including parity blocks), or stripes, across disks in the array distributing the parity blocks across the array as shown in Figure 1. For a read access smaller than the block length only one disk is accessed. For larger read accesses multiple disks are accessed. By distributing both the data and parity blocks the read throughput of the array is increased. This is achieved as the read/write requests are distributed evenly across each disk in the array, enabling parallel execution of disk accesses. For RAID 5 parity calculations, it is necessary to have a copy of all data in the stripe to perform the calculation; the stripe is therefore normally read into cache to maximize write throughput.

Figure 1 shows an example of a RAID 5 array with data stripes numbered D0-D7 and the parity stripes numbered P1–P4. This pattern of data and parity blocks is distributed across the disk to improve data access speeds. Write operations which do not access all the data disks require that the parity disk be updated to reflect the changes. This is done by subtracting the modulo 2 of the difference between the old data and new data from the parity stripe. RAID 5 distributes the parity throughout the array to lower the likelihood of the parity write access becoming a bottleneck. Multiple writes are therefore possible in a RAID 5 array across different stripes. RAID 5 is the most flexible of the RAID levels, offering good support for small write operations and high read and medium write data rates.

2.2 RAID 6

RAID level 6 is largely undefined and was not part of the original RAID definitions proposed in Patterson et al. [1988]. It is loosely defined as offering support for double disk recovery. This is achieved through a more complex algorithm than that used in RAID 5 and requires a minimum of two checksum disks. As in RAID 5, disk striping is used to improve the write performance. An example of a RAID 6 array with two data disks and two parity disks is given in Figure 2. Here the first checksum stripe is numbered P1–P4 and the second checksum stripe numbered Q1–4. RAID 6 requires both the checksum disks to be updated for every write operation decreasing

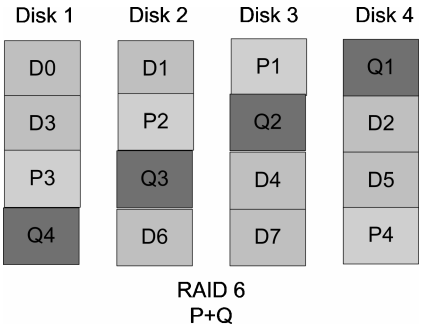

Fig. 2. RAID 6 implementation.

the maximum throughput achievable for write operations, but offering equivalent performance as RAID 5 for read operations.

RAID level 6 may be considered a solution for mission-critical applications, as well as being suited to larger disk arrays which have a higher probability of failure, especially when using lower-cost, lower-reliability disk drives. As data storage capacities increase in an array the probability of read errors occurring during recovery also increases. With a RAID 5 implementation any read error during the rebuilding of an array will result in permanent data loss. RAID 6 is able to recover this lost data in instances where a single drive has failed. Actual data loss will not occur unless read errors occur across multiple disks in the stripe, or until two simultaneous drive failures occur and a read error is encountered during data recovery. During normal operation RAID 6 requires a single extra write operation when compared with RAID 5, however, read throughput matches that of a RAID 5 implementation.

The disadvantages of this approach are the complexity of the controller and the necessity of n+2 drives to implement the design due to the two-dimensional parity scheme. RAID 6 remains nonstandardized and as such many vendor-specific RAID level 6 schemes exist. It has been suggested that standardization of RAID 6 may take place since Intel has taken an interest in RAID 6 [Karp 2007]. A number of algorithms have been proposed to support RAID 6 and these include the use of Reed-Solomon codes and orthogonal dual parity checks [Hennessy and Patterson 2002].

3. RAID 6 ALGORITHMS

The following section provides an overview of algorithms which have been proposed for use in RAID 6 implementations.

3.1 EVENODD

EVENODD was first proposed as a suitable method for tolerating up to two disk failures in a RAID system [Blaum et al. 1995]. The benefit of EVENODD is that the encoding scheme is achieved through the use of exclusive-OR operations meaning that it may be implemented through the use of a standard RAID 5 controller hardware. This scheme makes use of both horizontal and diagonal redundancy. The diagonal redundancy carries the parity information that may be either even or odd, which is where this scheme derives its name. It is optimal for two disk failures as it only requires two redundant disks, and small write operations only require 3 read and 3 write operations. The major disadvantage of this encoding scheme is that the write performance for certain diagonal blocks is poor, resulting in a degradation in performance of the system.

3.2 Redundancy Matrix 2 (RM2)

RM2 RAID 6 architectures place data and parity stripes according to a Redundancy Matrix (RM) [Park 1995]. Parity blocks are associated with groups of data disks to provide double disk redundancy. It provides the means to uniformly distribute parity data across disk arrays, across disk arrays, although by grouping disks in this manner more than two additional redundant disks may be required. The algorithm is not readily applicable in practice as it assumes that disks can be partitioned into any number of blocks. It is of note as it highlights the need to distribute parity data to maximize data write throughput and remove potential bottlenecks. Whilst optimizations of the basic approach have been proposed [Nam et al. 2002], RM2 remains complex to implement and is suboptimal in the number of redundant drives required.

3.3 Reed-Solomon Coding

Reed-Solomon (RS) codes are a class of error correcting codes which enable the recovery of any n errors from a system with $m + n$ data inputs [Reed and Solomon 1984]. For application to RAID 6 arrays the desired property is that any 2 storage devices may fail and the data can be recovered from the remaining disks. Reed-Solomon codes provide optimal error correction capability as they require only n redundant storage locations to recover n failures.

Reed-Solomon-based disk arrays may operate in a manner similar to RAID 5, block-interleaved distributed-parity disk arrays. The difference is the use of Reed-Solomon coding in place of a basic parity calculation used in RAID 5 and the need for two redundant disks and two parity writes per stripe. Reed-Solomon encoding and decoding is performed using finite field arithmetic which has hampered RS-RAID from widespread adoption in hardware RAID controllers due to the increased complexity of the controllers. The benefit of a Reed-Solomon encoding scheme for RAID controllers over the other possible algorithms is that they operate over a two-dimensional array structure, unlike the three-dimensional structure of EVENODD codes, and are very similar to RAID 5 arrays. RAID 5 may be considered as a Reed-Solomon implementation for an $m + 1$ recovery system.

3.4 Galois Field Theory

Reed-Solomon coding is based upon Galois field theory and the a basic understanding of the theory of Galois fields is helpful in understanding the nomenclature of RS codecs. A Galois field is a field of finite order or cardinality. A Galois field of q elements is usually denoted as $GF(q)$. The number of elements in a finite field must be of the form p^m, where p is a prime integer and m is a positive integer, and is denoted as $GF(p^m)$ or $GF(q)$. By giving its size, a field is described completely as for any q of the form p^m, the field is unique up to its isomorphism: that is, only fields of the power of the prime exist.

The order of an element α in a $GF(q)$ is the smallest positive integer m such that $\alpha m = 1$. An element with order $(q - 1)$ in $GF(q)$ is known as a primitive element in $GF(q)$. There is always at least one primitive element in a $GF(q)$. As the $(q - 1)$ consecutive powers of α must be distinct, they form the $(q - 1)$ nonzero elements of $GF(q)$ [How 2006].

3.4.1 Calculating Primitive Polynomial GF(4). For a $GF(4)$ the prime field is $GF(2)$ represented as binary 0 and 1. This is can be shown as $4 = 2^2$. All addition and multiplication operations performed in this field are performed modulo 2 making it ideal for hardware implementations. To find the irreducible polynomial of degree 2 in $GF[x]$ by

a brute-force method, list all the quadratic rings of degree 2 with the coefficients of 0 and 1.

x^2
$x^2 + 1$

The only irreducible polynomial in this list is $x^2 + 1$ as it does not factor to zero. This can be done again for $GF(8)$, $GF(16)$, etc., where the field is of type $GF(2^m)$. Our prime will always be $GF(2)$ and we need find the m polynomials with coefficients of 0 and 1 have to be checked to find the irreducible primitives.

3.4.2 Galois Field Arithmetic. The polynomial representation of a Galois field is commonly used for addition operations. Over a Galois field the associative and commutative laws apply for addition and multiplication operations. This means that both addition and subtraction are the same and may be performed by a bitwise XOR operation. These features allow Galois-field-based coding schemes to be readily implemented in hardware.

3.5 RAID 6 Implementations

In 2005, Intel released its IOP331 Xscale I/O processor which incorporated a hardware RAID 6 accelerator [Intel Corp. 2005]. Intel adopted a Reed-Solomon coding scheme for the RAID 6 checksum generation. The hardware acceleration units on the device assist in the calculation of parity and Galois field arithmetic. This controller is used in a number of commercial RAID controller cards to provide hardware RAID 6 solutions. In August 2006, Xilinx issued a white paper on the implementation of RAID 6 hardware acceleration for their FPGAs [Xilinx Inc. 2006].

Reed-Solomon codes appear to be the solution of preference for commercial hardware RAID 6 controllers at present. Software RAID 6 based upon a Reed-Solomon encoding scheme is also available as part of the standard Linux kernel and has been included since version 2.6 was released. This allowed for the hardware implementations that we describe shortly to be directly compared with a comparable software implementation.

4. THE DESIGN OF A RAID 6 CONTROLLER

In this section we present the first of our hardware RAID 6 implementations. This implementation is optimal for arrays of four disks and was designed to require minimal hardware resources. Whilst RAID 6 is not commonly used in such small disk arrays there are applications, such as CCTV recording and other security applications, where the ability to remove two drives and generate a direct replacement of all the data is of benefit.

All FPGA development presented this section was performed using Verilog HDL and simulated in Mentor Graphics ModelSim PE. Where possible system-level testbenches were written, again in Verilog HDL, to simulate the hardware design and also verify through built-in checks that the hardware operated as expected. The testbench verified the accelerators, ability to encode and decode all combinations of input data for all possible array combinations. Synthesis was undertaken using the Altera Quartus II design tools. Our PCI-based hardware was implemented on an Altera Stratix EP1S20F780 PCI-X/PCI Development Kit 32/64-bit from PLDA [PLDA 2002]. All other hardware implementations were implemented on an Altera Cyclone II development board.

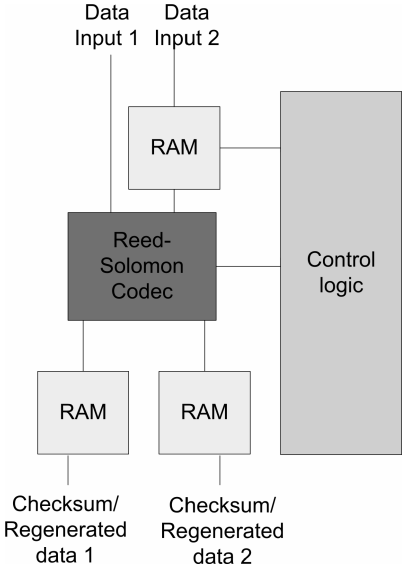

Fig. 3. Hardware implementation of the Reed-Solomon encoder using $GF(4)$.

4.1 Reed-Solomon Codec

The codec presented in this section operates in a $GF(4)$ and is the basis of an optimal RS-RAID 6 accelerator for arrays of four storage devices. The codec is optimal for this purpose as it can support only two data disks and two checksum disks.

The hardware codec is designed to operate under the following assumptions.

(1) The data disks and checksum disks are located at fixed locations, that is, data disks at location 1 and 2 and checksum at location 3 and 4.
(2) Both data inputs are passed into the codec simultaneously.
(3) 2 bits per input are encoded or decoded simultaneously.

The checksum is generated according to a generator function. Encoding of the first checksum is performed by a simple bitwise XOR operation of the two data inputs, both two bits wide. Generation of the second checksum required that each input be multiplied by a fixed value over the Galois field. For this codec the first input is multiplied by one and the second by two. These multiplicative operations are performed by a lookup table. The hardware implementation of the Reed-Solomon encoder is shown in Figure 3.

Figure 4 shows the path through the decoding hardware when both of the erasures occur in the data disks.

If one or both of the checksum disks are erased then the lost data is regenerated by passing the data through the Reed-Solomon encoder and selecting the appropriate output. If one or both of the erased disks contained data then the missing data is regenerated using the appropriate decoding functions as described in Eqs. (2) and (3).

$$\begin{aligned}(d_i \times i) \oplus (d_j \times j) &= c_1 \oplus s_2 \times i \\ \Rightarrow (d_i \times i) \oplus (d_j \times j) &= (c_2 \oplus s_2) \oplus [(c_1 \oplus s_1) \times i] \end{aligned} \tag{2}$$

$$(d_i \times i) \oplus (d_j \times j) = c_1 \oplus s_2 \Rightarrow \frac{(c_2 \oplus s_2) \oplus [(c_1 \oplus s_1) \times i]}{i \oplus j} \tag{3}$$

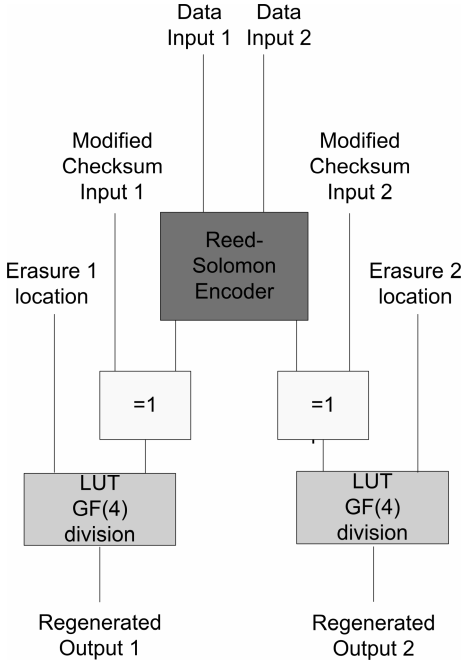

Fig. 4. Hardware implementation for the Reed-Solomon decoder using $GF(4)$ for double data disk recovery.

The RS codec operation was verified by encoding all combinations of two-bit inputs and storing the results. All single and double disk failures are tested using the stored data and verifying the output with the original data. This exhaustive test of the codec demonstrates that the encoder and decoder operate correctly for all possible data inputs, and recover from all permutations of single and double erasures.

4.2 RS RAID Hardware Accelerator

To demonstrate this RAID 6 implementation in a practical system and verify its performance a PCI interface was added to our design. This necessitated further control logic and packaging of our original implementation. Instantiating multiple instances of the codec in parallel allowed us to match the codec data input width to that of the PCI bus width. To control data flow and ensure data is processed correctly a memory interface is added to allow data to be buffered before and after encoding/decoding takes place. Additional control logic to control the flow of data into and out of the codecs and configure the encoding and decoding operations is also provided. A block diagram of the RS RAID accelerator is shown in Figure 5.

The memory buffers are implemented to enable the first block of input data from the first disk to be stored locally within the RAID accelerator block. The second disk's data is input to the codec whilst simultaneously reading the buffered data.

The hardware was tested by simulation for all combinations of double and single disk failures using the previously generated Verilog HDL testbenches. The RS codec was tested by encoding all combinations of two-bit inputs and storing the results. All single and double disk failures were then tested one at a time using the stored data and verifying the output with the original data. These showed the hardware accelerator was able to correctly encode and recover all combinations of double and single disc

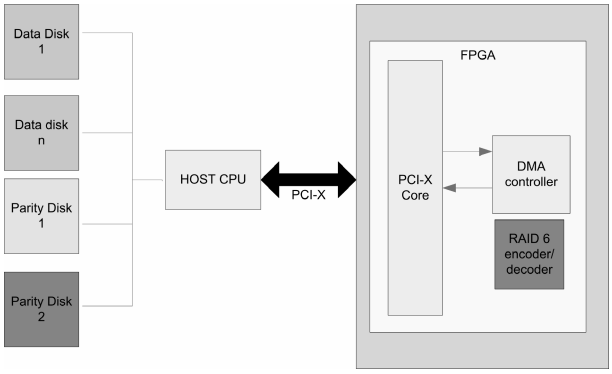

Fig. 5. RS RAID accelerator block diagram.

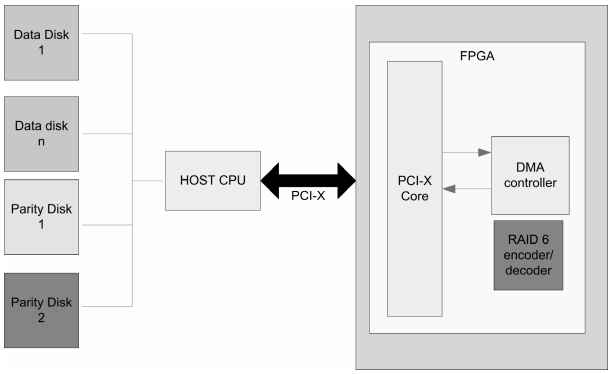

Fig. 6. PCI-based RAID 6 accelerator block diagram.

erasure. The results of timing analysis after synthesis for a Altera Cyclone II showed the accelerator capable of operating at a clock rate in excess of 200 MHz.

4.3 PCI RAID 6 Accelerator System Architecture

Having developed the RS RAID accelerator it was necessary to quantify the performance benefit achieved through the use of hardware acceleration. The accelerator block was implemented on an Altera Stratix FPGA-based PCI-X development board with a PCI 64-bit 66 MHz bus, DMA controller, and memory blocks. To enable rapid software driver development the software RAID 6 implementation of the Linux 2.6 kernel was modified to support the hardware accelerator. All comparisons in performance between the software RAID 6 and our hardware implementation were made using the unmodified kernel.

The test platform consisted of: a server class computer running Linux kernel 2.6.9.2 on an AMD Opteron processor with 1GB of RAM; 1 parallel ATA drive for the operating system; and four 200GB Maxtor DiamondMax Serial ATA drives for testing RAID 6 arrays attached via a four port SATA controller, Si3114, from Silicon Image [Maxtor Corporation 2005b; Silicon Image 2007].

To determine the read/write throughput during normal disk operations Bonnie++ [Bray 2006] was used to provide benchmark results for the test platform. Bonnie++ is a benchmarking tool used to determine data throughput performance for different types of read and write accesses on hard disk drives. To ensure that all the required

Table I. Benchmark Results for Test Platform Using Software RAID and Hardware RAID 6

Array Configuration	Sequential read throughput	Sequential write throughput
Single Disk	50.3 MB/sec	49.7 MB/sec
RAID 0	209 MB/sec	197 MB/sec
RAID 5	102.8 MB/sec	90.4 MB/sec
software RAID 6	68.4 MB/sec	59 MB/sec
Hardware RAID 6	59.5 MB/sec	59.3 MB/sec

data could not be cached, which may result in unrealistic performance metrics, a 4GB file size was used for read and write operations during benchmarking. A summary of the results of this benchmarking is given in Table I. All benchmark figures were generated using the unmodified Linux kernel and software RAID implementations.

Our test platform had a number of limitations for testing our RAID 6 accelerator. As we did not have direct attachment to the hard disk drives, data must be read from the drives by the software driver then passed to the RAID 6 hardware accelerator. The encoded/decoded data was then passed back to the CPU to then be written back to the hard drives. As a direct result of this limitation the maximum data throughput when using the hardware accelerator will be halved. Also, as the software driver for controlling access to the disk drives did not reside on the hardware accelerator we expect a further performance degradation compared to a complete stand-alone RAID 6 hardware controller.

5. RELIABILITY TESTING

5.1 Array Generation and Recovery

To verify that the hardware accelerator operated correctly within the test system a RAID 6 array was generated and verified using the Linux *mdadm* tools and the modified RAID 6 device driver. Using the hardware accelerator it was possible to successfully:

— generate a RAID 6 array;
— build a Linux file system;
— mount the RAID 6 array;
— read and write files to the array.

Verification of data write integrity has been conducted by comparing a known written file with its original and checking for any differences. In this and all other tests each of the four disk drives utilized 10GB partitions, offering 20GB of usable data storage space. The 10GB partitions were specifically selected to ensure that the test results were not skewed in favor of the software-only solution due to the 1GB of RAM available on the server and to enable the time taken to build and rebuild arrays to be kept to less than one hour.

With the RAID 6 array functioning correctly under normal operating conditions, an exhaustive test was carried out of all possible combinations of disk failure and recovery to verify the hardware simulation results. Each disk was removed and rebuilt separately; files are written to and read from the array whilst a single disk is removed (operating in the degraded mode) and during the rebuild stage. Once the array is rebuilt, the files on the array are compared with the original files to verify that the data has been successfully recovered. These steps are repeated with each permutation of double disk erasure. Again the array successfully regenerated the missing data.

Table II. Data Throughput for Software and 64-bit 66 MHz PCI Bus-Based Hardware Accelerator

	Software RAID 6	Hardware RAID 6
Array generation speed	17 MB/sec	23 MB/sec
Data recovery 1 erasure	17 MB/sec	23 MB/sec
Data recovery 2 erasure	15 MB/sec	25 MB/sec

Table III. Bonnie++ Results for Sequential Read and Write Operations Using the Hardware Raid 6 Accelerator and the Software RAID Implementation

	Sequential block read operations		*Sequential block write operations*	
	Throughput KB/sec	% CPU	Throughput KB/sec	% CPU
HW normal operation	59507	25	59347	7
SW normal operation	68427	14	59098	7
HW single erasure	29334	7	74910	10
SW single erasure	30734	7	92891	12
HW double erasure	33527	8	76531	12
SW double erasure	32884	7	67390	8

Direct comparison of the software RAID 6 solution with the hardware accelerated solution is shown in Table II. The performance benefit of the hardware-accelerator is not as significant as would be expected due to system architecture of having to access the hard disk drives via the PCI bus as discussed in Section 4.3.

5.2 Read/Write Performance

The read/write performance of both the hardware and software RAID 6 were benchmarked using Bonnie++. There was a clear improvement shown in using the hardware accelerator over the software-only solution when a double disk failure had occurred. A short extract of the results for both software and hardware RAID 6 is presented in Table III. It should be noted that the software RAID 6 was operating under ideal operating conditions. The hardware-accelerated system was using an inefficient architecture as the hard disk drives were not directly connected to the FPGA.

These additional bus transactions result in the hardware accelerator operating at least half as efficiently as would be expected in a full RAID 6 controller implementation. When migrated to a more appropriate RAID controller design it would be expected that the accelerator will provide at least twice the data throughput as that shown here.

The software driver performance was found inconsistent with the data throughput, dropping dramatically as disks failed, whereas the hardware performed consistently during all modes of operation. It was also noted the software was prone to periodic drops in performance for short periods which resulted in low data throughput rates. These drops in data throughput corresponded to increases in the CPU utilization of the test systems, background processes and would appear to demonstrate that the software performance was highly dependant upon the loading of the server. These drops in performance could be by as much as 90% of the average data throughput rates. This was not observed when using the hardware accelerator.

The hardware accelerator was able to produce a data throughput rate which closely matched those of the software solution under normal operation, offering a

slight gain during single disk failures and a greater benefit during double disk erasures.

5.3 CPU Utilization

The hardware accelerator appeared to be occupying large amount of CPU time when performing the Bonnie++ benchmarking tests. This was the opposite of what was expected. Upon closer examination of the processes running on the CPU during the testing it was found that this CPU utilization was actually being used by Bonnie++ and not the hardware device driver. During the software RAID 6 testing the majority of the CPU was utilized performing RAID 6 parity calculations and accesses. During hardware testing the majority of the CPU time was spent by Bonnie++ continually checking if the data had been processed or not. The actual CPU utilization of the hardware driver was found to be around 3–4% during all operational modes. The software driver CPU utilization was found to be around 33% higher during normal operation increasing to over 50% higher during double disk failures. These results may indicate that Bonnie++ was not the ideal tool to use to perform our benchmarking, however, it was the best solution available at the time.

6. HARDWARE RAID 6 FOR UP TO 15 DISK ARRAYS AND BEYOND

Having successfully demonstrated the hardware RAID 6 accelerator using a Reed-Solomon codec using $GF(4)$ arithmetic, it was necessary to produce an improved version which offered support for a larger number of disk drives. Two implementations are presented in this section. The first provides support for up to 15 disks in a single array and is the RS coding performed over a Galois Field of 16, $GF(16)$, using 4-bit wide inputs as opposed to the 2-bit wide inputs of the original codec. The target application in this instance was to support a stand-alone storage system providing dual disk redundancy. Larger disk arrays of up to 255 drives are supported when performing calculations over a Galois field of 256, $GF(256)$. In this instance a input bus width of 8 bits is used.

Where the original RS codec provided separate encoder and decoder blocks operating within strict limits for input orders, the new codecs were designed to be more flexible and accommodating. The codecs required the following additional information:

— number of disks in the array;
— offset of the first checksum;
— offset of the second checksum;
— current disk offset;
— location of disk erasures;
— modified checksum input data.

By providing this information the codecs are able to perform the checksum generation or data recovery automatically as the data path through the codec is altered based upon these inputs.

The improved codecs, instead of requiring all the input data to be buffered prior to performing the coding, encodes the input immediately. This requires that the results of previous disk inputs, the modified checksum, be input along with the new data input. A simplified block diagram showing this improved codec design for the is shown in Figure 7.

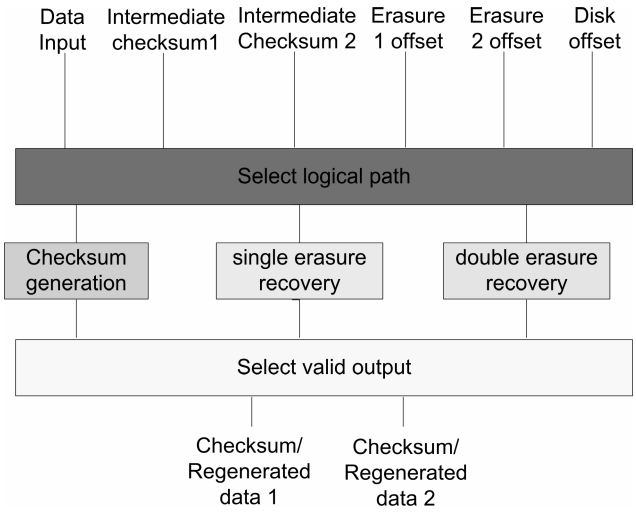

Fig. 7. $GF(16)$ or $GF(256)$ codec.

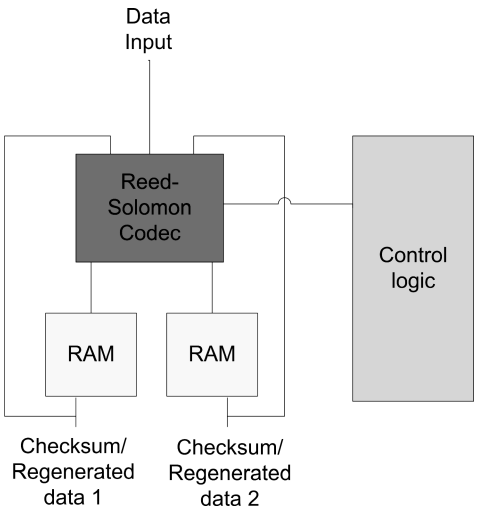

Fig. 8. $GF(16)$ RS RAID 6 hardware accelerator.

6.1 Modified RAID 6 Accelerator

To enable generation of the appropriate checksum for larger arrays when data is processed on a disk-by-disk basis the intermediate checksum which is generated as each individual disk is processed must be stored for use in processing the next disk in the stripe. To ensure data throughput performance this intermediate checksum was stored locally and this memory block was configured to automatically output the previously encoded data into the codec as new data arrived, as illustrated in Figure 8. This design change freed up memory resources consumed by the previous RAID 6 accelerator used to buffer the input prior to performing coding operations. Again multiple instances of the codec were instantiated in parallel to support 32-bit and 64-bit coding operations.

Table IV. Resource Utilization of the Hardware Raid 6 Accelerator Blocks

	$GF(4)$ codec	$GF(16)$ codec	$GF(256)$ codec
LUT Utilisation per RS codec	63	304	1243
LUT Utilisation for RS RAID 6 accelerator	841	4854	7105
Maximum clock rates on a Stratix FPGA	204.7 MHz	198.7 MHz	197.6 MHz
Coding latency	3 clock cycles	7 clock cycles	8 clock cycles

6.2 Comparison of RAID 6 Accelerators

The $GF(4)$-based Reed-Solomon codec has been optimized to take advantage of a fixed structure for data to be input and output. This enables minimization of the logic resources required for the $GF(4)$ codec. To reduce the overheads associated with configuring the $GF(16)$ and $GF(256)$ codecs and to make it more flexible it has been designed to be agnostic to: the input sequence, the number of disks in the array (provided there are no more than 15), and offer greater flexibility than available from the original codec. The overall resource utilization and maximum clock rates for both codecs are detailed in Table IV.

Despite the $GF(16)$ codec being designed for flexibility rather than optimized for hardware resource consumption like the original codec, the resource utilization of the $GF(16)$ codec was 20% of that demonstrated by the only other known published implementation [Song et al. 2005]. The codec may also be clocked at a higher rate with a lower clock latency than this area-efficient architecture.

The modified RS RAID 6 hardware accelerators were added seamlessly into the existing PCI-based test platform. The software driver was modified to provide the additional configuration data required by these codecs.

An additional Silicon Image 3114 SATA controller card was added to provide access to a fifth SATA disk drive. Verification of a five-disk RAID 6 array was also performed at this point. The differences in performance for the hardware-accelerated design and the unmodified software RAID 6 solution were equivalent to those found using an array of four disks. This shows that the performance of the hardware RAID 6 codec continues to perform as expected.

Larger disk arrays were not benchmarked due to a lack of suitable SATA disk drives and a lack of space in the test system to add additional drives. However, by partitioning the five disk drives which were available, larger arrays were shown to work with the modified RAID 6 hardware accelerator.

7. RAID 6 IP BLOCK FOR AVALON BUS

To enable rapid verification of the $GF(16)$ and $GF(256)$ codecs in hardware an Avalon bus wrapper was added to the design. The benefits of implementing the RAID 6 accelerator on the Avalon architecture was that it enabled testing and performance metrics to be gathered on the RAID 6 accelerator for all combinations of disk array. For a more generic implementation the PCI control was stripped from the DMA controller and an Avalon bus wrapper was added to the IP block. The Avalon bus based RAID 6 accelerator is shown in Figure 9.

7.1 Avalon Test System

A testbench was used to verify basic Avalon bus transactions. However, as the SOPC builder enabled rapid system design and development it was decided that

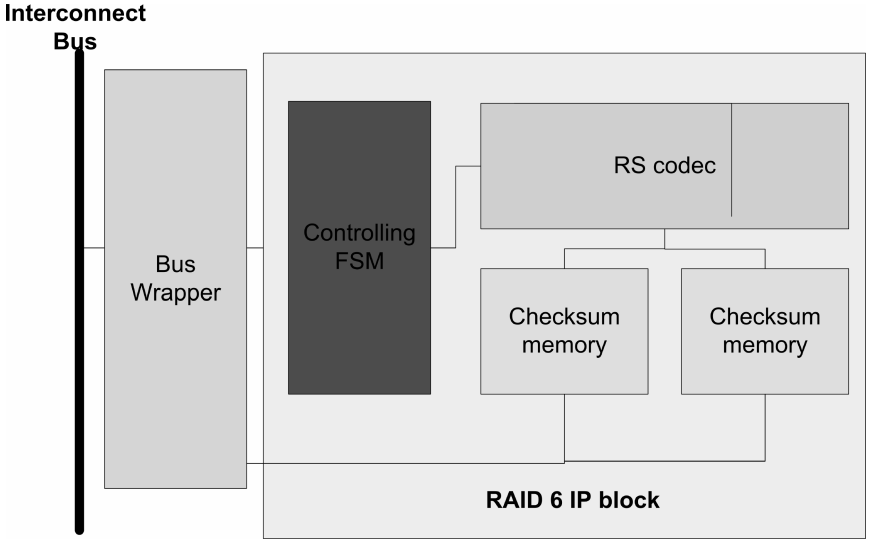

Fig. 9. Avalon bus-based RAID 6 IP block.

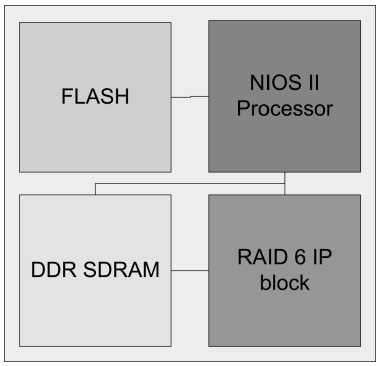

Fig. 10. Avalon bus-based test system on the cyclone II development board.

the generation of a simple Avalon bus-based system under the control of a NIOS II processor would produce more rapid and valid test results. Using the Cyclone II development kit an example system was set up with access to the following.

— NIOS II processor: Provides overall system control and a simple and rapid means to generate test data and verify the returned data.
— Serial EEPROM: Used to store the software programs for the NIOS.
— DDR SDRAM interface: Connected to 64MB of DDR RAM to be used as program memory.
— RAID 6 IP block.

This hardware configuration is shown in Figure 10. These four blocks were connected via the Avalon bus using the SOPC builder and a simple test system was synthesized.

Software was developed for the NIOS II processor using the Alteras NIOS II development kit with the source code written in C. A program was developed which segmented regions of the DDR RAM to act as four disks in a RAID 6 array. Data was

written to two of these disks and the RAID 6 hardware accelerator was configured to generate the checksums for the remainder. Once the RAID 6 array was successfully generated, the data disks were marked as having suffered an erasure and the data regenerated from the checksum. This regenerated data was checked with the original data to verify that the data had been recovered correctly.

This testing was repeated for all combinations of single and double disk erasure and for larger disk arrays. In every instance the hardware RAID 6 accelerator successfully regenerated the lost data and wrote it back to the correct locations.

8. CONCLUSION

We have developed and demonstrated three Reed-Solomon-based RAID 6 controllers, each designed to address separate target applications. Our RAID 6 accelerators are readily interchangeable to enable the optimal implementation to be selected for a given application. The first provided an optimal implementation for arrays of four disks. The second RAID 6 accelerator provides support for up to 15 disks in an array. The results of testing this design has shown that it requires only 20% of the physical resources of the previous best implementation as well as operating at higher clock rate. This enables our hardware accelerator to provide a higher data throughput than previously shown. Our final implementation provided support for the more common $GF(256)$-based RS RAID 6 controller. The scalability of the $GF(256)$ codec has made it the implementation of choice for commercial hardware RAID 6 implementations. From our investigation we have demonstrated that significant reductions in hardware resources may be made through the adoption of the algorithm most appropriate to the target application. At the same time we have sustained the same data throughput for each separate codec.

ACKNOWLEDGMENTS

The authors would like to thank the EPSRC for their financial support. They would also like to thank A2E Ltd. for their financial and technical support throughout the project.

REFERENCES

BLAUM, M., BRADY, J., BRUCK, J., AND MENON, J. 1995. Evenodd: An efficient scheme for tolerating double disk failures in raid architectures. *IEEE Trans. Comput. 44,* 2, 192–202.

BRAY, T. 2006. *BONNIE++*. http://code.google.com/p/bonnie-64/.

CHEN, P. M., LEE, E. K., GIBSON, G. A., KATZ, R. H., AND PATTERSON, D. A. 1994. RAID: High-Performance, reliable secondary storage. *ACM Comput. Surv. 26,* 2, 145–185.

HENNESSY, J. L. AND PATTERSON, D. A. 2002. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Publishers, San Francisco, CA.

HOW. 2006. *Fields and Galois Theory*. Springer.

INTEL CORP. 2005. *Intel 80331 I/O Processor Datasheet*. Intel Corp.

KARP, M. 2007. All about raid. *Netw. World News.*

MAXTOR CORPORATION. 2005a. *Atlas 15K II SAS datasheet*. Maxtor Corporation.

MAXTOR CORPORATION. 2005b. *DiamondMax 10 datasheet*. Maxtor Corporation.

NAM, Y. J., KIM, D.-W., CHOE, T.-Y., AND PARK, C. 2002. Enhancing write i/o performance of disk array rm2 tolerating double disk failures. In *Proceedings of the International Conference on Parallel Processing (ICPP'02)*. IEEE Computer Society, Los Alamitos, CA, 211.

PARK, C.-I. 1995. Efficient placement of parity and data to tolerate two disk failures in disk array systems. *IEEE Trans. Paral. Distrib. Syst. 6,* 11, 1177–1184.

PATTERSON, D. A., GIBSON, G., AND KATZ, R. H. 1988. A case for redundant arrays of inexpensive disks (raid). In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. ACM Press, New York, 109–116.

PLDA. 2002. *Advanced Altera Stratix PCI-X/PCI 32/64-bit Development Kit*. PLD Applications.

537 REED, S. AND SOLOMON, G. 1984. Polynomial codes over certain finite fields. *SIAM J. Appl. Math. 8*,
538 300–304.

539 SILICON IMAGE. 2007. SiI3114 PCI to serial ATA controller datasheet. Silicon Image.

540 SONG, M.-A., LAN, I.-F., AND KUO, S.-Y. 2005. An area-efficient architecture of reed-solomon codec for
541 advanced raid systems. In *Proceedings of the 11th International Conference on Parallel and Distributed*
542 *Systems (ICPADS'05)*. IEEE Computer Society, Los Alamitos, CA, 620–626.

543 XILINX INC. 2006. *Hardware Accelerator for RAID6 Parity Generation/Data Recovery Controller*. Xilinx Inc.

544 Received October 2008; revised November 2009; accepted January 2010